MATch: Differentiable Material Graphs for Procedural Material Capture

LIANG SHI, MIT CSAIL
BEICHEN LI, MIT CSAIL
MILOŠ HAŠAN, Adobe Research
KALYAN SUNKAVALLI, Adobe Research
TAMY BOUBEKEUR, Adobe
RADOMIR MECH, Adobe Research
WOJCIECH MATUSIK, MIT CSAIL

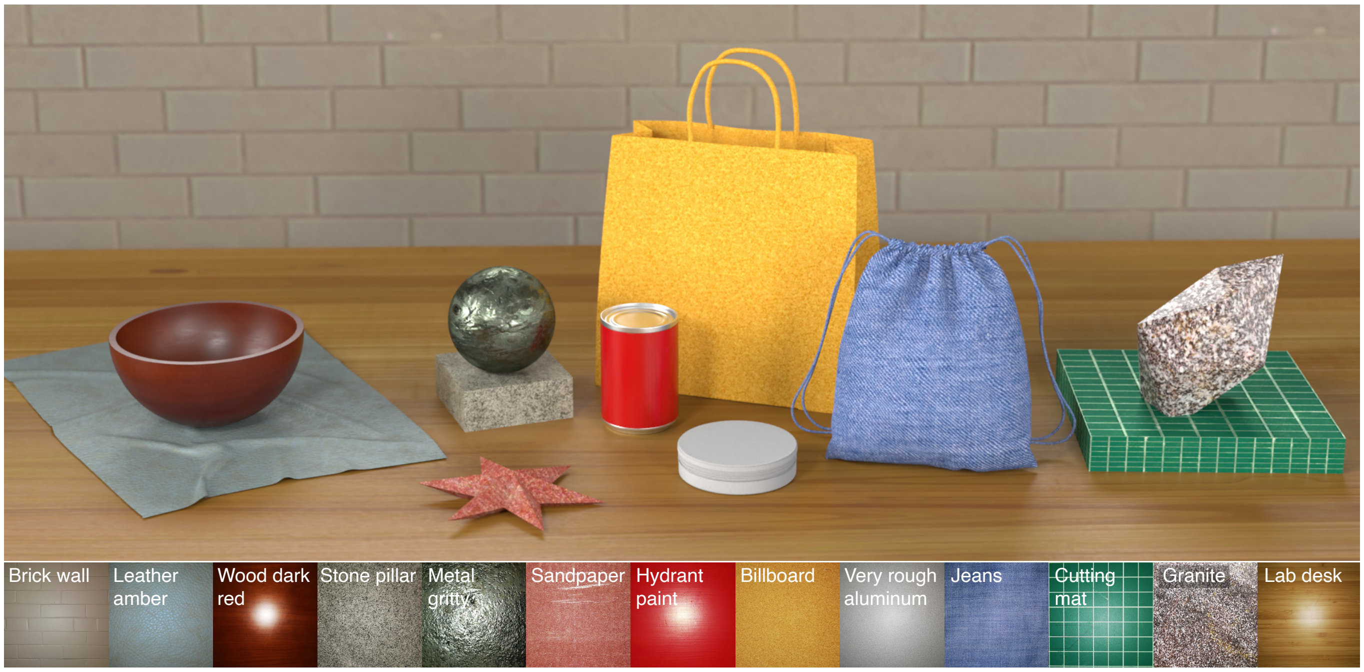

Fig. 1. Every material in this rendered scene is a procedural material that was automatically created by MATch from a single flash photograph captured with a cellphone. All the target materials are visualized underneath. The created materials can be found in Figure 6 and Supplementary Material Figure S2.

We present *MATch*, a method to automatically convert photographs of material samples into production-grade procedural material models. At the core of MATch is a new library *DiffMat* that provides differentiable building blocks for constructing procedural materials, and automatic translation of large-scale procedural models, with hundreds to thousands of node parameters, into differentiable node graphs. Combining these translated node graphs with a rendering layer yields an end-to-end differentiable pipeline that maps node graph parameters to rendered images. This facilitates the use of gradient-based optimization to estimate the parameters such that the resulting material, when rendered, matches the target image appearance, as quantified by a style transfer loss. In addition, we propose a deep neural feature-based graph selection and parameter initialization method that efficiently scales to a large number of procedural graphs. We evaluate our method on both rendered synthetic materials and real materials captured as flash photographs. We demonstrate that MATch can reconstruct more accurate, general, and complex procedural materials compared to the state-of-the-art. Moreover, by producing a procedural output, we unlock capabilities such as constructing arbitrary-resolution material maps and parametrically editing the material appearance.

CCS Concepts: • **Computing methodologies → Rendering**.

Additional Key Words and Phrases: procedural materials, material acquisition

ACM Reference Format:
Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. 2020. MATch: Differentiable Material

Authors' addresses: Liang Shi, MIT CSAIL, liangs@mit.edu; Beichen Li, MIT CSAIL, beichen@mit.edu; Miloš Hašan, Adobe Research, mihasan@adobe.com; Kalyan Sunkavalli, Adobe Research, sunkaval@adobe.com; Tamy Boubekeur, Adobe, boubek@adobe.com; Radomir Mech, Adobe Research, rmech@adobe.com; Wojciech Matusik, MIT CSAIL, wojciech@csail.mit.edu.

0730-0301/2020/12-ART196
https://doi.org/10.1145/3414685.3417781

Graphs for Procedural Material Capture. *ACM Trans. Graph.* 39, 6, Article 196 (December 2020), 15 pages. https://doi.org/10.1145/3414685.3417781

1 INTRODUCTION

Procedural materials have become very popular in the computer graphics industry (movies, video games, architecture, and product visualization). These materials are often represented as node graphs, where each node may denote simple image processing operations, but the collective graph can produce material maps (like albedo, normals, roughness, etc.) for highly complex, real-world spatially-varying BRDFs (SVBRDFs). In contrast to SVBRDFs that are explicitly defined in terms of per-pixel material parameter maps, such procedural material models have a number of advantages: they are compact in memory, resolution-independent, efficient to evaluated for interactive feedback during the material design process, and can be easily edited to generate material variations. However, such procedural materials are manually designed by expert artists using professional tools—a time-consuming process that is often well beyond the capabilities of novice users.

The goal of our work is to enable a light-weight material acquisition method that can automatically create procedural materials from captured images. While this would be extremely challenging to do from scratch, there already exist datasets of high-quality procedural materials[1] that can serve as a starting point. Therefore, we present MATch (short for MATerial Match), a method to convert a target RGB image (for example, but not limited to, a flash photograph taken with a cellphone) into a procedural material by identifying an appropriate procedural model from a model library, and estimating the node parameters of that model to best match the target appearance. In particular, we focus on estimating the continuous node parameters in the filter nodes of the procedural model, while keeping the discrete parameters and generator nodes fixed (these typically govern the types and random seeds of base patterns and noises). As we show in this paper—and illustrate in Figure 1 where every single material has been automatically captured from a single cellphone flash photograph—this is already sufficient to reproduce a wide variety of real-world materials.

MATch is an *optimization-based* method where we treat the node graph as analogous to a neural network whose parameters can be estimated; in our case the parameters are not network weights, but the various node parameters in the graph. This approach requires a key element: a differentiable version of a given material graph that can be evaluated forward and backward (for gradient computation). We accomplish this by introducing a procedural material modeling library DiffMat to translate procedural material node graphs into differentiable programs. We demonstrate that DiffMat can handle production-grade procedural material graphs (with hundreds of nodes and thousands of parameters) representing complex spatially-varying BRDFs.

We combine the translated differentiable material graphs with a differentiable rendering layer to create an end-to-end differentiable graph parameter-to-image pipeline that enables parameter estimation through gradient descent. Inspired by previous work on texture synthesis and image style transfer [Gatys et al. 2015; Gatys et al. 2016], we use losses based on the statistics of deep neural features of the VGG network [Simonyan and Zisserman 2015] to compare the rendered image to a target example [Aittala et al. 2016; Guo et al. 2019]. Optimizing the graph parameters using these losses allows us to reproduce the target image appearance without requiring perfect pixel alignment between these images.

We apply our graph translation scheme to publicly available procedural material libraries[1] to create a set of 88 differentiable graphs. Given an example RGB image, we propose an efficient graph selection scheme to first identify the most appropriate graphs out of this set. We use a pre-trained VGG network to extract deep features from the image and use graph-specific shallow networks to predict parameters from these extracted features. These network-predicted parameters are coarse but sufficient to select the top-3 graphs that best match the input image appearance. Finally, we optimize for the parameters of the selected graphs using our differentiable pipeline.

Together, our three contributions—**(i)** An efficient graph selection and node parameter initialization network, **(ii)** an optimization scheme and loss function that recovers accurate parameter values from captured images, and **(iii)** a procedural material modeling library DiffMat that can automatically translate procedural graphs into differentiable node graphs—enable single-shot high-quality procedural material capture. Compared to the state-of-the-art inverse procedural material design work of Hu et al. [2019] who also select a procedural material from a library and estimate its parameter to match an input image (albeit via direct network prediction as against our optimization-based approach), we recover additional BRDF parameters that define complex, real-world materials (such as roughness and metallicity), estimate a significantly larger set of node parameters for production-scale procedural models and demonstrate more accurate, photorealistic reconstructions. Unlike recent work on single-image material capture that reconstructs per-pixel spatially-varying BRDFs [Aittala et al. 2016, 2015; Deschaintre et al. 2018, 2020; Li et al. 2018b], which are limited in resolution and surface coverage or require an initialization from a resolution-limited method [Gao et al. 2019], our method converts images into procedural materials that are higher-quality, resolution-independent, editable, support seamless tiling and have small storage requirements. We illustrate this in Figure 1 by constructing procedural materials from a wide range of real-world materials captured using a hand-held cellphone in unconstrained conditions.

2 RELATED WORK

Material Capture. Capturing the parameters of spatially-varying BRDFs is a classic problem in computer graphics; please see Guarnera et al. [2016] for a recent survey of this body of work. While one could scan the entire 6-dimensional SVBRDF using a spherical gantry, this is expensive and rarely necessary. The goal of many previous methods is to reduce the measurement effort and output parameter maps for diffuse color, normal, specular roughness/glossiness, and specular albedo. Among the first solutions that scanned such maps using a linear light was Gardner et al. [2003]. More recently, Aittala et al. [2013] used Fourier patterns projected

[1]https://share.substance3d.com/, https://source.substance3d.com/

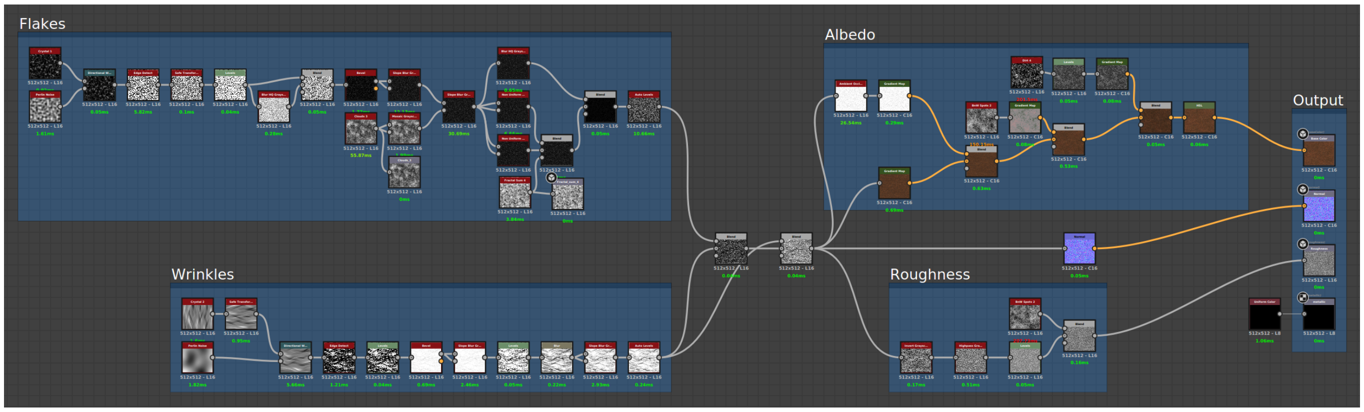

Fig. 2. An exemplar leather material graph created using Substance Designer. The graph starts with several generator nodes that provide initial patterns for subsequent filter nodes to manipulate. The graph is internally partitioned into subgraphs that create different visual attributes of the leather pattern. It outputs four material maps that define the parameters of an SVBRDF.

onto an LCD screen in combination with a general optimization algorithm to recover per-pixel SVBRDF parameters, but their number of measurements was still quite high (hundreds).

Aittala et al. later reduced the number of measurements required to two photos [2015] and later a single photo [2016], under the assumption of stationarity (perceptual uniformity) of the output textures. A key component of the latter work was a texture descriptor, inspired by work on texture synthesis and style transfer work [Gatys et al. 2015; Gatys et al. 2016], and based on feature maps of the VGG network [Simonyan and Zisserman 2015]. We also use this approach with some modifications, both in our optimization and in training predictor networks.

More recently, the work of Li et al. [2018b] and Deschaintre et al. [2018] introduced a direct end-to-end prediction approach using convolutional neural networks, mapping single-image measurements to material parameter maps. These approaches have also been extended to handle multiple input images, either via pooling of features [Deschaintre et al. 2019] or inverse rendering-based optimization in an auto-encoder latent space [Gao et al. 2019]. For large-scale materials, Deschaintre et al. [2020] fine-tuned the network on close-up flash photographs of the test sample to achieve better generalization for the entire material. While these results can be impressive, the reconstruction quality varies with the input, often showing blurring or loss of quality for novel views or light positions. Moreover, the resulting maps have a low resolution (typically 256×256), are not tile-able nor easily editable afterwards. For these reasons, procedural outputs have significant advantages. In fact, using a procedural material model can be thought of as applying a "prior" to the underconstrained material capture problem; this has a regularizing effect that, in addition to the other advantages of a procedural model, also leads to higher-quality results.

Procedural noise by example. Several previous methods produce noise patterns that can be controlled by example images. Galerne et al. [2012] produce noise textures with specified power spectra based on Gabor noise. The parameters controlling the noise characteristics can be estimated from examples. More recently, Galerne et al. [2017] have introduced texton noise, a related approach of higher performance, while Heitz and Neyret [2018] developed a fast histogram blending approach to produce non-repeating noise-like textures from small exemplars. While these approaches are related to our work, as they are also driving a procedural model by (real or synthetic) example images, they are limited to relatively simple stationary noise patterns. On the other hand, they have advantages such as speed and small memory footprint.

Inverse Procedural Material Modeling. Hu et al. [2019] recently introduced a method for inverse procedural material modeling that, like us, given an input RGB image, selects a procedural material graph from a library, and estimates graph parameters to best match the given image. However, unlike our optimization-based approach, they train neural networks to directly predict the parameters from input images. In their work, they constrain themselves to a Lambertian BRDF model (predicting only diffuse albedo and surface normals) and predict only a subset of the graph parameters (the "exposed" parameters of the graph). Moreover, these network predictions can be coarse, leading to apparent differences between the input image and the predicted graph result. They propose using a post-process style transfer step to better match the results. Unfortunately, this is a non-parametric step that diminishes the advantages of procedural representation. Finally, their work requires a separate deep parameter prediction network for every single graph in their library, leading to a substantial memory footprint. On the other hand, our pipeline is fully differentiable and refines the prediction by iterative optimization. This allows us to optimize for all node parameters of a graph and support non-Lambertian BRDF models, which leads to significantly more accurate material reconstructions. Finally, our network selection and parameter prediction use deep features from a pretrained network and only train shallow fully-connected networks per graph, making it much more compact. We believe our work is a major evolution of their framework, with higher flexibility and power.

The recent work by Guo et al. [2019] also addresses parameter estimation for procedural material models. Their focus is more on a Hamiltonian Monte Carlo sampling approach in a Bayesian framework. They also explore style transfer losses based on the Gram matrices of VGG features. However, their procedural materials are small hand-written pieces of code, and their approach does not utilize network parameter prediction nor automatic material choice. In contrast, we substantially expand the set of materials to closely match the capabilities of production-scale systems, and scale to many more estimated parameters (hundreds), while also combining optimization with neural network prediction.

3 PROCEDURAL MATERIALS: OVERVIEW AND DIFFERENTIABILITY

Procedural material node graphs are directed acyclic graphs of largely two types of nodes: *generators* and *filters*. Generator nodes create spatial textures from scratch based on user-specific parameters, and include both noise generators (like Perlin noise) and structured pattern generators. Filter nodes manipulate input textures using operations ranging from pixel value manipulations (like color or contrast edits) to image processing (like filtering, warping, blending, etc.); these nodes are parameterized by the control parameters of the functions they implement (for example, kernel size for a box filter or opacity for a blending node). The connectivity of the graph defines a sequence of operations that start with 2D scalar or vector maps (usually created by generator or data store nodes) and manipulate them (usually operations defined by the filter nodes) to finally output a set of materials maps. In this work, we focus on four material maps—albedo (or base color), normals, roughness, and metallicity—that specify the parameters of the simplified Disney BRDF model [Burley 2012] that has become the de-facto standard in the real-time and offline rendering industry.

Procedural materials have several desirable properties. They are often resolution-independent—changing the resolution of the generator outputs changes the final texture map resolution. Editing the various node parameters generates different material variations. Procedural design-based tools also enable artists to hierarchically design materials. For example, an artist can construct a graph that produces a specific material appearance and embed into a larger graph to generate more complex materials. Thus, while the basic nodes of a procedural material design tool might be simple operations, they can be combined to construct significantly more complex nodes or sub-graphs that can be reused in many material designs.

Figure 2 shows an example leather procedural material graph. This graph has 43 filter nodes and 127 node parameters with fairly complex connectivity that allows it to produce a wide range of photorealistic leather material appearances. Typical production procedural materials are even more complex than this example.

3.1 Differentiability of Node Graphs

Recent work on learning-based inverse graphics has found that combining neural networks with graphics models can lead to more interpretable, and in many cases, more accurate results [Deschaintre et al. 2018; Hu et al. 2018; Li et al. 2018b; Tewari et al. 2017]. However, this requires that the graphics model itself be differentiable to allow for training via back-propagation. This has prompted the development of differentiable libraries for computer vision [Riba et al. 2020], rendering [Hiroharu et al. 2018; Li et al. 2018a; Loper and Black 2014; Nimier-David et al. 2019], geometry processing [Fey and Lenssen 2019] and spatially sparse computing [Hu et al. 2020]. Similarly, we wish to explicitly use procedural node graphs as the output of material capture as they provide a compact and interpretable material representation. This is currently not possible because there exist no tools to efficiently evaluate procedural materials (forward and especially backward) in an optimization framework.

Our insight is that typical procedural material *filter nodes* consist of image processing operations that are closely analogous to operations common in convolutional neural networks (CNNs). With very few exceptions, filter nodes either act as convolutions (blurs, edge detectors, etc.) or as per-pixel operations (curves, blends, thresholds) to each pixel of the input map(s) separately. This is closely related to the typical components of CNNs: convolutions and point-wise operations (activation functions such as ReLU, sigmoids, etc.). This leads to our insight that the filter nodes in a production procedural material system can, in fact, be fully translated to a modern machine learning framework, and thus being fully differentiable.

On the other hand, the *generator nodes* are not always expressible in this way. However, in this work, we choose to not optimize generator node parameters and focus only on estimating the filter node parameters. The reason is that changing generator node parameters tends to produce different instances of the same pattern (for example, different instances of the same kind of noise); that is, the parameters act mostly as random seeds. We consider these variations to be different instances of the *same* material (for example, corresponding to different pieces of the same leather material). Since our goal is to reproduce the visual appearance of the target image and not the exact pixel values (which would be extremely challenging if not impossible), we find that optimization of generator parameters is not typically necessary. In contrast, changing the filter node parameters, often fundamentally changes the material appearance (for example, from matte brown leather to glossy red leather).

Based on these observations, we implement a procedural material modeling library DiffMat consists of differentiable atomic (base-level) filter nodes and compound (high-level) filter nodes based on the atomic set. DiffMat also integrates an automatic graph translator that translates complex procedural material graphs into differentiable graphs. In Sec. 4, we describe how we use these translated differentiable graphs to produce high-quality materials from input images. In Sec. 5, we discuss the implementation of our differentiable material graph library.

4 MATERIAL CAPTURE WITH PROCEDURAL GRAPHS

Given a library of differentiable graphs, we follow a two-step process to recover a procedural material from a target photograph. First, we identify the most relevant graphs for the input photograph. We train networks to, given an image, directly predict the node parameters of each graph. We process the input photograph with these parameter prediction networks, evaluate the error between images rendered with these predictions and the input image, and pick the top-3 graphs with the lowest errors. Second, we use an optimization-based method to fit the parameters of each of these

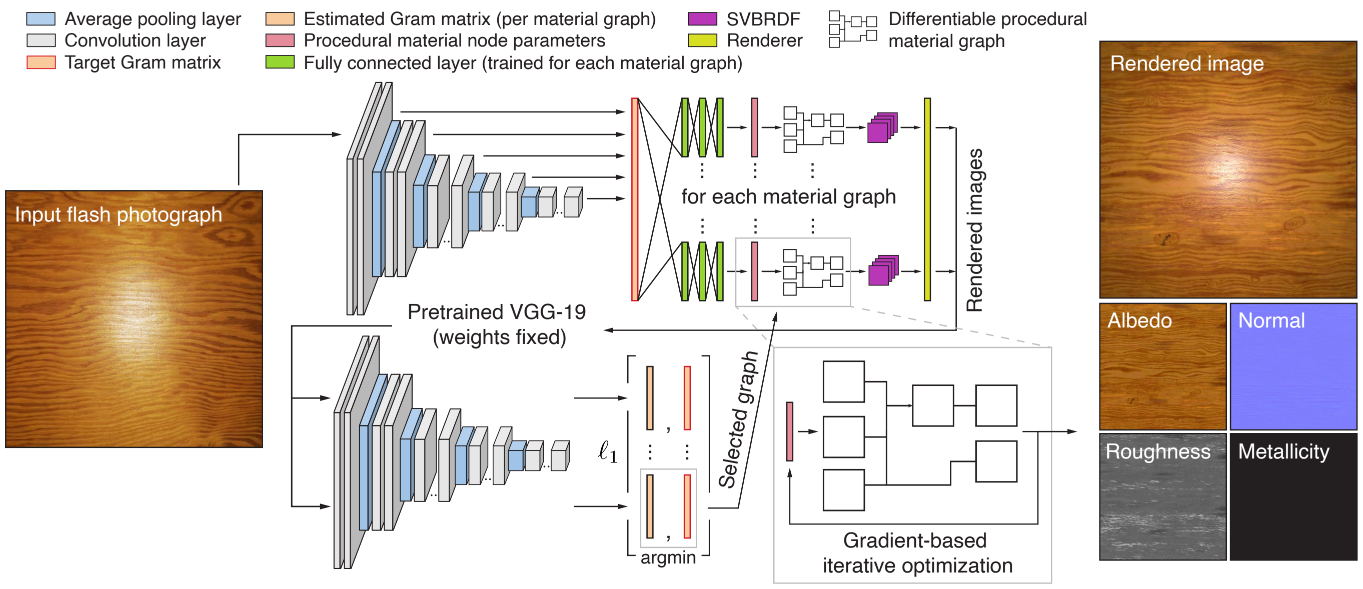
Average pooling layer
Convolution layer
Target Gram matrix
Estimated Gram matrix (per material graph)
Procedural material node parameters
Fully connected layer (trained for each material graph)
SVBRDF
Renderer
Differentiable procedural material graph
Input flash photograph
Pretrained VGG-19 (weights fixed)
for each material graph
Rendered images
Selected graph
ℓ_1
argmin
Gradient-based iterative optimization
Rendered image
Albedo
Normal
Roughness
Metallicity

Fig. 3. Overview of MATch's fully differentiable procedural material design pipeline. Given an input target image (potentially a flash photograph captured using a cellphone), we compute an input texture descriptor (flattened Gram matrix) using a pre-trained VGG network. For each procedural material in our library, we train a parameter prediction network by appending fully connected layers to this texture descriptor. We use these models to predict node parameters of the target image for every material graph, and generate the corresponding SVBRDFs using the differentiable material graphs translated by DiffMat. We render images from these SVBRDFs, compute their corresponding texture descriptors and select the top-3 material graphs with the lowest ℓ_1 texture descriptor difference with respect to the input. Finally, we refine the previously predicted node parameters using a gradient-based optimization with our differential node graphs to improve the match with the target image. The material graph (along with the refined node parameters) that produces the closest matched result is output as the final result. Here, we visualize this pipeline with a real, captured wood material sample.

graphs to the target image. Our entire pipeline is illustrated in Figure 3. We now describe this process in detail, starting with our parameter optimization method.

4.1 Image-based Parameter Optimization

Given a specific translated graph $\mathcal{G}$, our goal is to estimate node parameters that will produce a spatially-varying BRDF whose rendered appearance will reproduce a target image, $\boldsymbol{I}^*$. In particular, we are interested in the material parameter vector $\boldsymbol{\theta}$ of length k, a concatenation of all k optimizable parameters of the node graph $\mathcal{G}$.

We define the parameter map evaluation operator M that encompasses the evaluation of $\mathcal{G}$. Given $\boldsymbol{\theta}$ as input, it produces parameter maps of a simple BRDF model combining a microfacet and diffuse term: albedo $\boldsymbol{a}$, normal vector $\boldsymbol{n}$, roughness $\boldsymbol{r}$ and metallicity $\boldsymbol{m}$ (the latter is a spatially-varying weight blending between a dielectric and metallic interpretation of the BRDF). Therefore,

$$(\boldsymbol{a}, \boldsymbol{n}, \boldsymbol{r}, \boldsymbol{m}) = M(\boldsymbol{\theta}). \tag{1}$$

As mentioned in Sec. 3, we do not optimize over generator node parameters (typically random noise seeds) $\boldsymbol{z}$. Instead, we precompute the generator outputs and keep them fixed in our estimation, i.e., we optimize $M(\boldsymbol{\theta}|\boldsymbol{z})$. In the following, we skip $\boldsymbol{z}$ for brevity.

The rendering operator R takes the generated maps and computes a rendered image under known illumination. We assume a single target image captured by centered co-located point light and camera (simulating a cellphone camera with flash), though extensions to other configurations, including multiple lights or views, is straightforward. The predicted synthetic image can be written as:

$$\boldsymbol{I} = R(M(\boldsymbol{\theta})) = R(\boldsymbol{a}, \boldsymbol{n}, \boldsymbol{r}, \boldsymbol{m}). \tag{2}$$

Note that both operators M and R are differentiable; this allows for gradient computation by backpropagating through the entire expression $R(M(\boldsymbol{\theta}))$. Differentiable rendering operators have also been previously used for material capture, albeit with per-pixel SVBRDF representations [Deschaintre et al. 2018; Li et al. 2018b].

Optimizing for $\boldsymbol{\theta}$ requires us to define a loss function between the rendered image, $\boldsymbol{I}$ and target image, $\boldsymbol{I}^*$. Such a loss cannot rely on pixel-perfect alignment of texture features because the spatial patterns between the two images are unlikely to match exactly. We use the popular style loss function that was proposed by Gatys et al. [2015; 2016] for image style transfer, and has been used by many previous methods, including material capture [Aittala et al. 2016; Guo et al. 2019]. This loss function can be written as:

$$L_G = \|T_G(\boldsymbol{I}) - T_G(\boldsymbol{I}^*)\|_1, \tag{3}$$

where T_G is a Gatys texture descriptor defined by the concatenation of the Gram matrices of the five feature maps before each pooling layer of the VGG [Simonyan and Zisserman 2015] network.

We compute the Gram matrix using the VGG-19 network without batch normalization, provided by the *torchvision* library. We also replace the max-pooling by the average pooling, as suggested by Gatys et al. We normalize images by the mean/variance for ImageNet

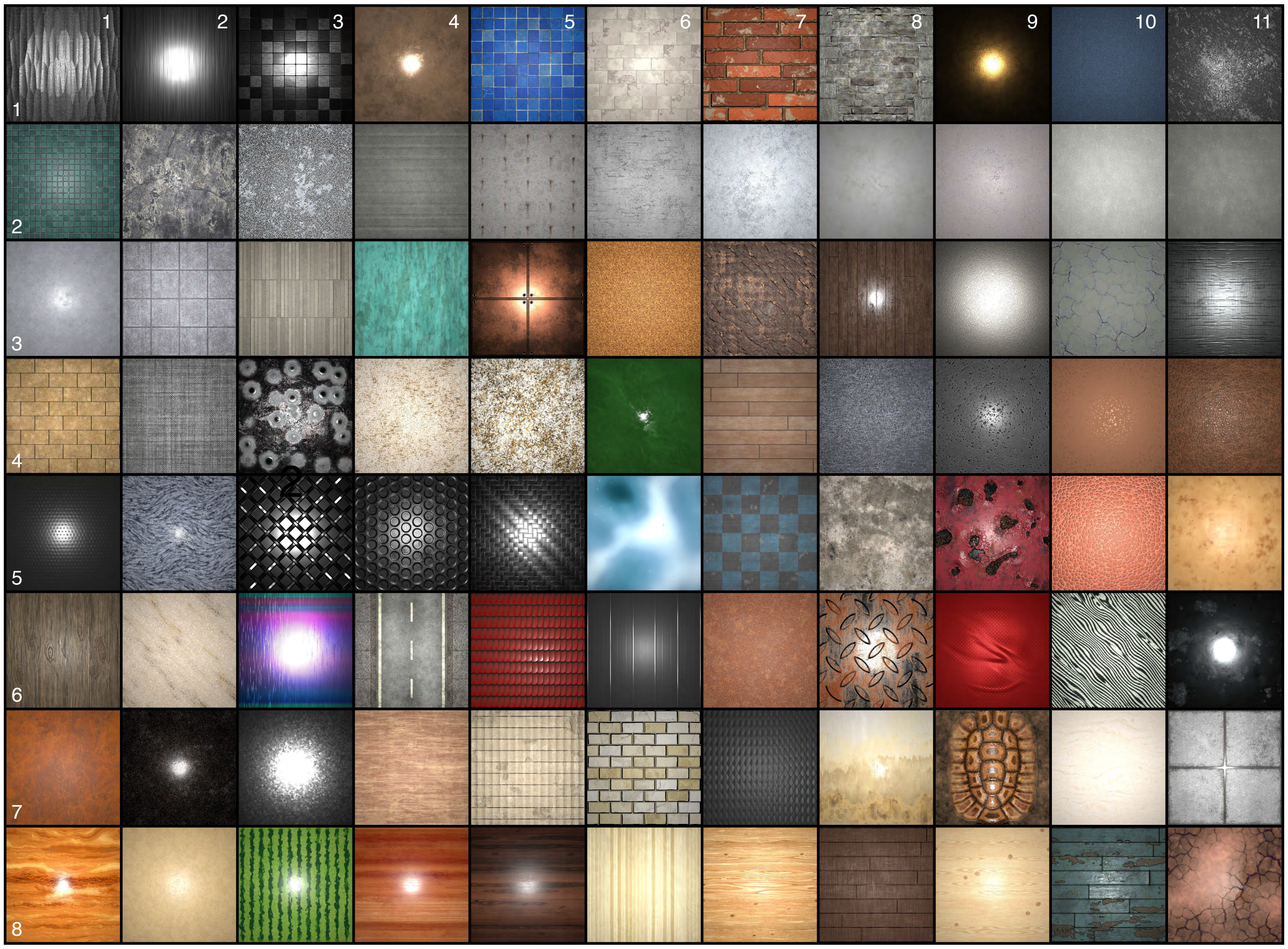

Fig. 4. Rendered images of 88 automatically translated procedural material graphs, marked with row and column index. All images are rendered from the maps produced by DiffMat. The image resolution is 512 × 512 pixels. Readers are encouraged to zoom in to examine image details.

data before feeding them to the VGG network. We compute the Gram matrix descriptors for images at multiple resolutions (specifically resolutions of 512, 256, and 128); this improves the perceptual match over multiple scales.

We use Adam optimizer [Kingma and Ba 2014] with a learning rate of 5e-4 to optimize the Gram matrix loss. Because Eqn. 2 is fully differentiable, we can backpropagate through the rendering and the whole material graph to update the values of $\boldsymbol{\theta}$ directly. This is in contrast to previous work that used the style transfer loss to optimize for material maps/images [Aittala et al. 2016].

4.2 Network-based Graph Selection and Initialization

The choice of the graph used to match a target image affects the quality of reproduction. Yet, when a large set of material graphs is given, manually choosing the appropriate graph is non-trivial; this is especially so, because as shown in Fig. 6, production-grade procedural graphs are often "over-parameterized" and can represent a wide range of material appearance beyond their semantic class. Therefore, we propose an automatic graph selection method. This method first directly predicts the parameters of each graph for the target image and uses this prediction to select candidate graphs. These predictions can also be used as initializations to the optimization, accelerating convergence, and leading to a better solution for difficult input targets.

Similar to Hu et al. [2019], we predict the material parameter vector using a neural network that takes the target image $\boldsymbol{I}^*$ and returns an estimate of $\boldsymbol{\theta}$. Hu et al. train a parameter prediction network, based on the AlexNet [Krizhevsky et al. 2012] backbone, for *every* procedural graph in their dataset. This incurs a significant effort to train models for each material completely from scratch and has a large storage footprint.

Instead of training individual deep networks end-to-end, we extract image features using a pretrained VGG network and only train multi-layer perceptrons with three fully connected layers for each procedural material. As shown in Figure 3, we construct the same Gram matrix descriptor T_G as above, which (when flattened) produces a vector of length l. We pass this feature through three fully connected layers of size $l \times 3k$, $3k \times 3k$ and $3k \times k$, with a ReLU unit in between the layers, where k is the number of graph parameters. A final sigmoid produces estimates in the range $[0, 1]$, which are then remapped to each parameter's actual range. Note that because the length of T_G is invariant to the input image resolution, the same architecture can be applied to varying input resolutions. This approach combines the advantages of a powerful feature extractor with the small compute and memory requirements of training small per-graph networks.

We train the fully connected layer weights with synthetic data generated at training time; thus no dataset storage nor I/O cost is incurred. For each iteration, we randomly generate a ground truth material vector θ^* and evaluate the corresponding maps $\boldsymbol{a}^*, \boldsymbol{n}^*, \boldsymbol{r}^*, \boldsymbol{m}^*$ (Eqn. 1) and the ground truth synthetic image $\boldsymbol{I}^*$ (Eqn. 2). We then pass $\boldsymbol{I}^*$ through the (fixed) Gram matrix descriptor T_G and the (trainable) fully connected layers, resulting in an estimated parameter vector θ. Because of our fully differentiable pipeline, we can generate the material maps and rendered images for this parameter vector, resulting in predicted maps $\boldsymbol{a}, \boldsymbol{n}, \boldsymbol{r}, \boldsymbol{m}$ and image $\boldsymbol{I}$.

We train the weights of the fully connected layers with a weighted combination of a parameter loss and style transfer loss (Eqn. 3):

$$L_{\text{net}} = \lambda_\theta L_\theta + \lambda_g L_G. \tag{4}$$

The parameter loss L_θ is defined as the ℓ_1 difference between material parameter vectors:

$$L_\theta = \|\theta - \theta^*\|_1. \tag{5}$$

The style transfer loss serves as an image-based regularizer and teaches the network how the predicted parameters influence the visual appearance of the output material. We note that this term is not available to Hu et al. [2019] due to their non-differentiable material graph. We have also experimented with a per-pixel ℓ_2 loss on the images and maps, but found the predicted parameters tend to produce blurrier maps.

We use Adam optimizer with a learning rate of 1e-4 and a time-based learning decay rate of 0.97 to train the network. We set $\lambda_\theta = 1.0$ and $\lambda_g = 0.0$ for the first 1000 iterations, and then update $\lambda_g = 0.05$ for the rest of training. We adopt this approach as we find the regularization of the style transfer loss is more effective when the estimated material appearance becomes reasonably close to the target. The number of required iterations varies depending on the complexity of the material graph, but we found 10000 iterations with a batch size of 5 is generally sufficient for most of the material graphs we tried. As we show in Figure 5, combining a shared high-quality VGG feature extractor with a small number of per-material layers yields high accuracy prediction at lower training footprints.

Given a user-input image, we extract its texture descriptor feature and feed it to the parameter prediction networks for every graph. The predicted parameters are then fed to their material graphs and rendered as images. We select the top-3 graphs whose renderings have the lowest error in the Gatys texture descriptors with respect to the input image. Finally, we optimize all these graphs (Sec. 4.1) to match the target image—starting from the predicted parameter as initialization—and keep the result with the lowest error.

While our graph selection and initialization approach has similarities to Hu et al. [2019], it also has several advantages. First, our graph selection and parameter prediction reuse the same network, whereas Hu et al. uses pre-trained VGG-19 and clustering for graph selection, and individual deep networks for parameter prediction. Second, when a new material graph is added to the material graph set, Hu et al. would require re-running the clustering with *all* existing material graphs; our approach only operates on the *new* material and does not require re-computing any existing networks. Finally, we select the material graph from the entire pool of materials, while Hu et al. only select from the graphs made for the matched materials. As we show later, selecting from all materials is beneficial, allowing us to use graphs intended for very different materials to successfully reproduce the input image.

5 DIFFMAT: A DIFFERENTIABLE PROCEDURAL MATERIAL LIBRARY

DiffMat is a PyTorch-based library whose goal is to support the functionality and expressiveness of production-grade procedural modeling tools while allowing for easy integration with deep learning tools. To this end, we use Substance Designer[2] as a reference and build DiffMat to provide differentiable node routines that can match it with per-pixel accuracy. We do this for two reasons: first, Substance Designer is a professional material authoring tool used widely in the graphics industry and it shares the same design pattern with other procedural material authoring tools. Thus, by matching its functionality, our framework can be integrated into real-world procedural material design workflows. Second, there exist large public libraries of high-quality artist-designed Substance Designer material graphs, that we leverage in our material capture framework. That said, our implementation is independent of Substance Designer, and can be adapted to match the capabilities of other node graph authoring applications. DiffMat will be released for non-commercial academic research use.

5.1 DiffMat Design Overview

Given the similarity between a material node graph—that sequentially processes inputs using filter nodes—and a neural network, we follow the general design of PyTorch in DiffMat's API design. Specifically, DiffMat defines function routines (similar to *torch.nn.functional*) for stateless evaluation of node operations, and wrapper classes as their optimizable equivalent (similar to *torch.nn*), whose internal attributes represent node parameters that can be optimized.

We consider continuous parameters (e.g., opacity in the blend node) as optimizable parameters and discrete parameters (e.g., number of tiles along the row in the tile generator node) as non-optimizable parameters. The wrapper class holds a trainable parameter list and initial values for continuous parameters. Calling the wrapper class with required input images and discrete parameters will evaluate its functional counterpart. Because different continuous parameters have different ranges, we store them in the wrapper class as trainable parameters with a range of $[0, 1]$ along with non-trainable,

[2]https://www.substance3d.com/products/substance-designer/

node-specific minimum/maximum parameters that are used to map the parameters to their final values. To prevent the optimization from driving a trainable parameter out of its domain, the wrapper class applies clamping internally before calling the forward function. Implementing the nodes in PyTorch gives us the gradient computation for free by using PyTorch's auto-differentiation.

DiffMat defines a base class *DiffMatGraphModule* for all optimizable graphs to inherit. A child class derived from *DiffMatGraphModule* calls its parent's initialization function to initialize all node classes used in the current graph, and then defines the actual graph structure in its own forward function. The base class also collects all derived nodes' parameters as its own parameter attributes to enable convenient graph-level optimization and provides helper functions such as trained variable export.

5.2 Generator vs. Filter Nodes

As mentioned before, Substance Designer node graphs usually consist of generator nodes, that produce texture patterns from scratch, and filter nodes, that manipulate these generated textures to create the final material maps. Instead of including the generator nodes in the differentiable material graph, we precompute their outputs using the Substance Designer Automation Toolkit [Adobe 2019] by randomly sampling the node parameters and save these textures as inputs for the rest of the graph. Generator nodes can have scale and offset parameters and we achieve the same effects with an affine transformation filter node. While this simplification means that we cannot optimize the generator node parameters, as we illustrate in Figure 6, we are still able to represent a very wide range of materials. Moreover, this avoids implementing a large set of generators that are computationally expensive to evaluate, predict and optimize. In the following, we detail the filter nodes supported by DiffMat.

5.3 Atomic Filter Nodes

Atomic filter nodes are the basic building blocks of any procedural material design tool. In other words, any graph when decomposed into its core operations, will consist of only these nodes. These include standard image processing operators such as blur, warp, blend, distance, gradient map, and color adjustments, but also procedural modeling-specific operations like *Fx-map* (Sec. 5.5) and *pixel-processor* (Sec. 5.6). DiffMat provides implementations of all 21 atomic filter nodes supported by Substance Designer and accurately reproduces their behavior. As noted earlier, we turn the atomic generator nodes (Bitmap, SVG, and Text) into precomputed inputs. A full list of implemented atomic nodes can be found in the supplemental material.

5.4 Non-atomic Filter Nodes

Non-atomic filter nodes are pre-made graph instances (compound nodes) that are constructed using atomic filter nodes and are designed to reproduce specific visual effects and enable high-level artistic control over material appearance. Substance Designer has more than 150 non-atomic filter nodes. DiffMat implements the 110 most frequently used non-atomic filter nodes in the Substance Source dataset and is thus able to represent a significant portion of these graphs. We fuse inefficient calls of atomic filter nodes in these pre-made graphs to improve their computational efficiency in DiffMat's implementation. A full list of DiffMat's non-atomic filter nodes can be found in the supplemental material.

5.5 *Fx-maps* and Tiling

The *Fx-map* is a node unique to Substance Designer that allows users to subdivide and replicate an image repeatedly while applying rotations, translations and blending. It is commonly used to create repetitive or fractal patterns, such as tiles, stripes, and various types of noises. As with other generator nodes, we do not implement a full-fledged *Fx-map* in DiffMat, though it is feasible in future work. Nevertheless, we implement a limited version of the node that can represent all "tiling"-based non-atomic nodes. Our implementation allows users to tile a single input image in a specified manner to form an output image that is continuous both inside and over its edges. It also supports stochastic generation by allowing variations in position, rotation, scaling and blending opacity.

5.6 *Value/Pixel Processor* and Exposed Parameters

The *value-processor* and *pixel-processor* allow users to define custom mathematical functions using predefined mathematical operators. While the *value-processor* applies a function to a single value, the *pixel-processor* applies the function to every input image pixel.

The *value-processor* is used to create *exposed parameters* for Substance Designer graphs. Production-grade materials graphs can have hundreds of node parameters, making it challenging for users to explore the space of material variations from the graph. To make this process more intuitive, many graphs have *exposed parameters*—a smaller set of artist-defined parameters, that give users high-level control over the material's variations. These *exposed parameters* are internally converted to individual node parameters using *value-processor* functions. For example, an "age" parameter in a wood graph can control the number of annual rings and be used to set all ring-related node parameters using *value-processors*.

The *pixel-processor* provides shader-like control to manipulate pixel values and can create advanced effects like half-toning. Most *pixel-processors* in Substance Source use pre-defined, deterministic mathematical operators. Hence, we implement the *pixel-processor* as a fixed node with non-optimizable parameters (similarly for the *value-processor*).

5.7 Automatic Graph Translation

Production-grade graphs that produce complex, real-world materials can easily have tens to hundreds of nodes with hundreds to thousands of parameters (see Supplementary Material Tab. 1 for details of a set of Substance graphs). Moreover, while some graphs have well-organized hierarchies and proper use of sub-graphs, we found that this is often not the case. This makes the manual conversion of a given procedural graph to PyTorch code cumbersome and error-prone. Instead, DiffMat implements an automatic graph translation tool that makes this process effortless and scales to large-scale graphs.

A Substance Designer document encodes the procedural material graph and its associated exposed parameters in XML. The DiffMat translator parses and analyzes this XML document and generates

Python programs that replicate the functionality of the graph. This translation is performed as follows:

(1) **Graph replication**. The translator first detects and stores the default values of all exposed parameters of the input graph. It then replicates the whole graph in DiffMat by using the corresponding PyTorch operations with the same connectivity. As described in Sec. 5.2, graph nodes without input connections are regarded as generators and converted to precomputed input bitmaps in the forward evaluation routine.
(2) **Node parameter conversion**. For every filter node, the translator interprets and converts its parameters according to a series of node type-specific rules. If a node parameter is defined as a dynamic function of one or more exposed parameters, its initial value is set by constructing and evaluating its *value-processor* using the pre-stored exposed parameters.
(3) **Data flow analysis**. After constructing the graph structure, the translator examines node contributions to the graph outputs to eliminate redundancy. While a graph may produce many material maps, we only consider four (albedo, normals, roughness, and metallicity). Therefore, the translator runs a backward breadth-first search (BFS) and a forward BFS to identify the largest subgraph that is accessible from these four output nodes and trims all unused nodes.
(4) **Program generation**. Finally, the translator performs topological sorting on the final graph and generates a node sequence consistent with data dependency. Based on this sequence, the translator organizes the results from previous steps and outputs a complete Python program.

For node graphs that use unimplemented filter nodes, the translator automatically replaces them with *passthrough* nodes. To enable effortless high-level post-editing, we offer the option to retain the exposed parameters during graph translation. In the subsequent optimization, users can choose to optimize either exposed parameters alone or exposed parameters plus the rest of node parameters not directly tied to the exposed parameters. If users wish to explore a larger design space, they can further choose to remove the *value-processors* prescribed by the exposed parameters and optimize all node parameters freely. This is often useful if users wish to create their custom exposed parameters after the optimization.

In Supplementary Material, we thoroughly evaluate DiffMat performance in terms of graph translation, SVBRDF reproduction accuracy (comparing to the reference results computed by Substance Designer), computational efficiency, and memory cost.

6 RESULTS

We analyze the performance of MATch on a wide set of synthetic images and captured real-world photographs. All results are computed from a single 512 × 512 input photograph captured (or rendered) under flash illumination. For all these results, we use "Default" to denote rendering of materials that created by the out-of-the-box, originally-designed node parameters of a graph, and "Target" to denote the target image whose appearance we would like to reproduce. Unless otherwise specified, all these results predicted/optimized against *all* the filter node parameters of the graph. We encourage readers to zoom into the figures to evaluate the visual quality of our results. We also include more results and experiments in the supplementary material and video.

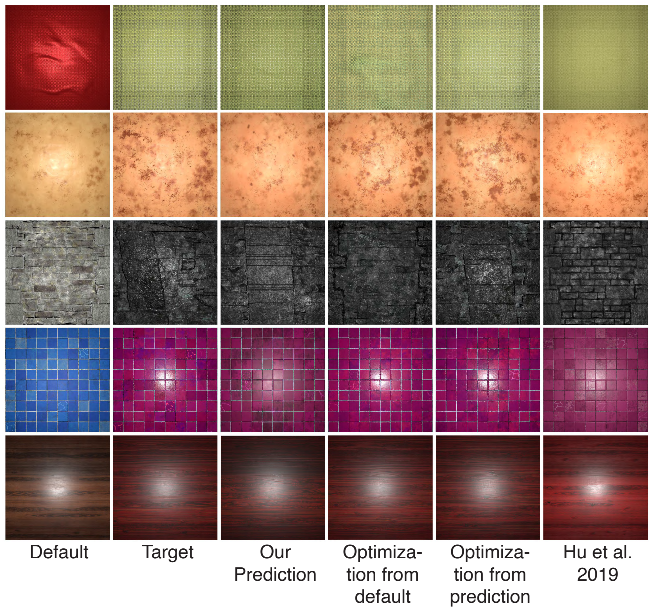

Fig. 5. Optimization and prediction results for synthetic materials. For a majority of synthetic examples, optimization alone is sufficient to reproduce the target materials. Compared to Hu et al. [2019] (our reimplementation on our data and BRDF model), our network prediction produces more accurate color, fine features, and glossiness with the help of rendering loss. The prediction result is further improved by the optimization step to precisely match the target material.

6.1 Material Capture from Synthetic Images

We first examine our approach on synthetic photographs. In this experiment, we manually sample the parameters of a graph to create new materials. We then render these materials and use the images to evaluate our direct prediction networks and the MATch optimization-based approach. In Figure 5, we show optimization only, prediction only, and prediction plus optimization results for 5 different material graphs. For a majority of examples, optimization alone (i.e., starting from the default parameters) is sufficient to closely reproduce target appearances. Direct prediction tends to coarsely match the target appearance, though in many cases misses the details that optimization is able to recover. That said, for challenging examples such as the black bricks, optimization from the predicted parameters improves reproduction quality over optimization from the default parameters. We emphasize that during both optimization and prediction, we deliberately use *different* noise patterns between the rendered images. This can be observed in the details of the rendered images. Yet, our style-based loss successfully handles this misalignment between the basic structures to accurately reproduce the overall material appearance. In the supplementary material, we visualize additional synthetic examples and the optimized SVBRDF maps.

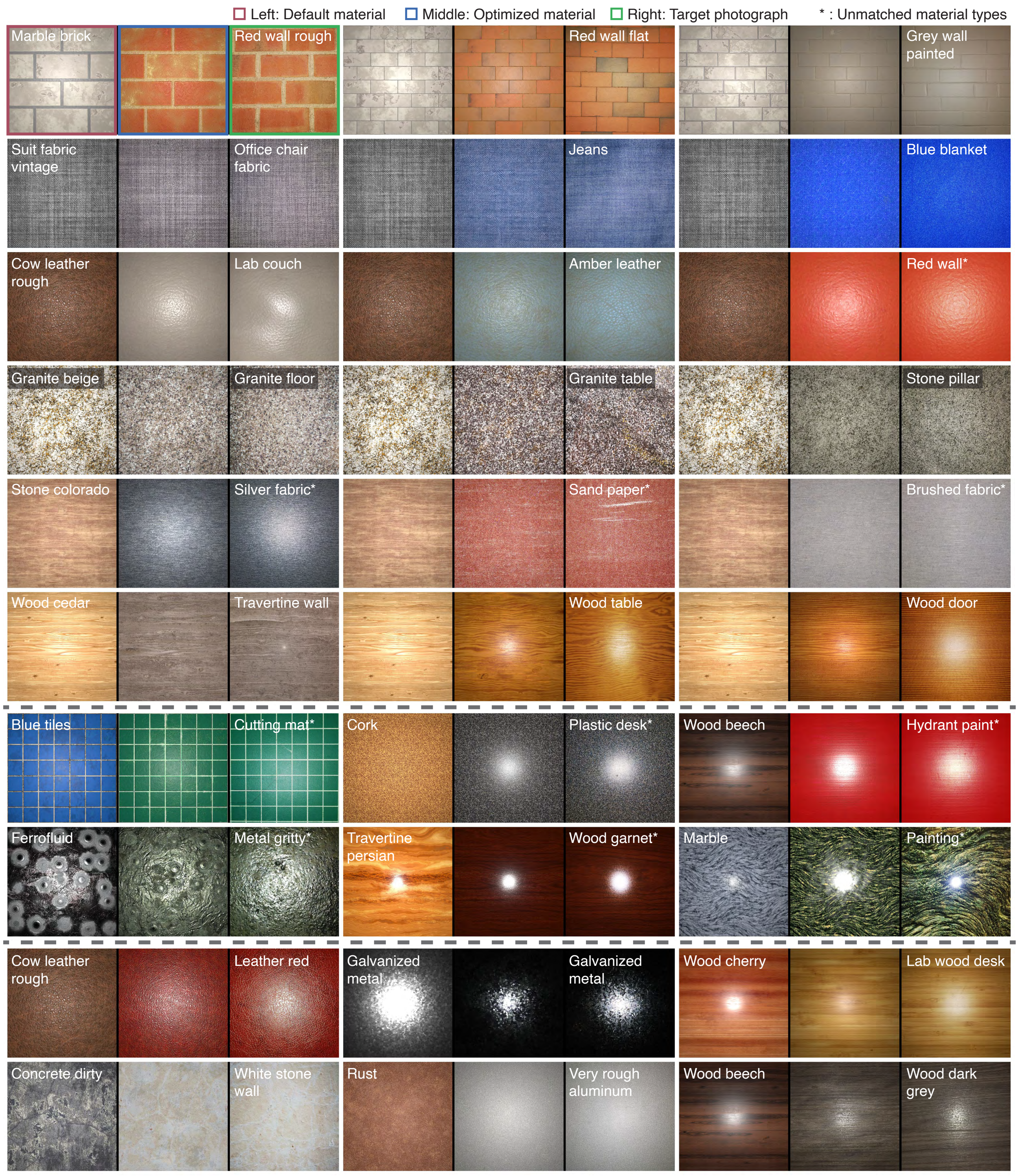

Fig. 6. Optimization results for real-world materials. **Top**: At each row, we show multiple real-world materials matched by one single procedural material graph using the same default initial parameters. The * symbol marks the examples that are optimized against a target material of a different material type. **Middle**: Additional examples on unmatched material and graphs. **Bottom**: Additional examples on matched material and graphs.

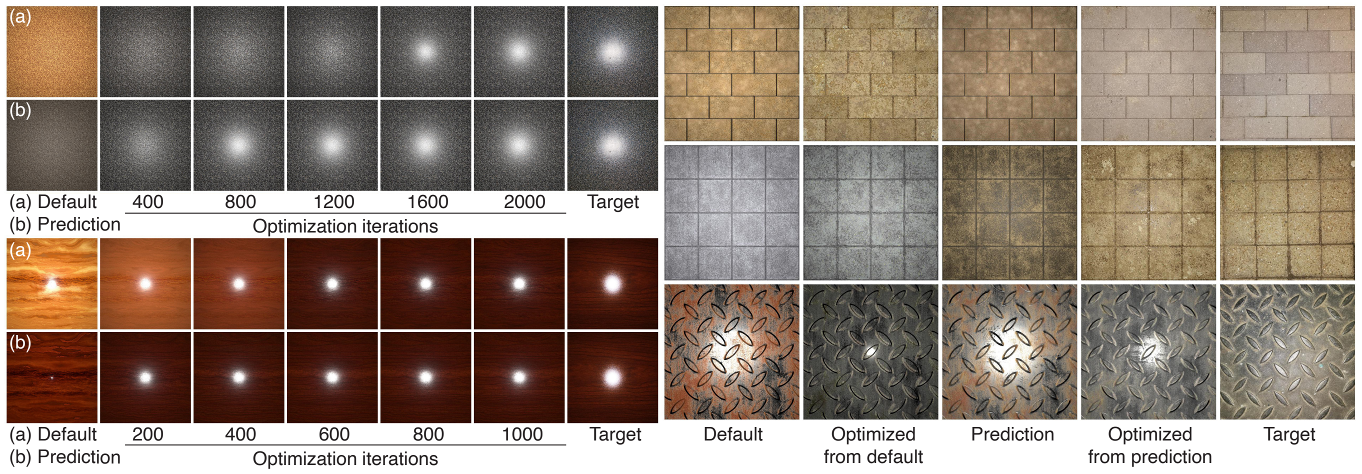

Fig. 7. Prediction plus optimization results for real-world materials. **Left**: Speed of convergence comparison between optimizing default and predicted node parameters. **Right**: Comparison of optimization results on challenging target materials . Optimizing from the default parameters stuck in the local minimum, while optimizing from the predicted parameters produces a significantly closer reproduction.

6.2 Material Capture from Real Data

Optimization for Real-world Materials. In Figure 6, we evaluate MATch on 30 real photographs with 27 images captured by us and 3 images from the dataset of Aittala et al. [2016]. In the supplementary material, we visualize additional real-world examples (44 in total) and the optimized SVBRDF maps.

We use the prediction network to pick the material graphs , but use the default parameters as initialization to demonstrate the capability of optimization. We note that being able to reproduce high-quality materials using just optimization allows users to quickly try out manually picked material graphs and skip the time-consuming network training step. Figure 6 (top) shows that we are able to reproduce a wide variety of materials starting from the same graph and initialization. This indicates the diversity of materials that can be generated by manipulating all the parameters in large-scale graphs, as well as the ability of our optimization-based approach to converge to a good solution.

In Figure 6 (middle), we show that when the chosen graphs are designed for a material type quite different from the targets, they are often expressive enough to represent the target materials (and the ability of the optimization to robustly discover these matches).

Prediction plus Optimization for Real-world Materials. While we find that optimization alone is sufficient for many examples, the prediction network generally provides a better initial guess of the node parameters, which accelerates the convergence of optimization and improves the final results. We demonstrate both these cases in Figure 7 on real-world materials. On average, the predicted parameters reduce the required iterations for convergence by more than half.

6.3 Comparisons with Previous Work

Comparison to Hu et al. In Figure 5 and Figure 9, we compare our results to the state-of-the-art procedural material capture method of Hu et al. [2019]. Their original work uses a different set of material graphs, assumes Lambertian BRDFs captured under natural illumination, and only predicts the exposed parameters of the graph. For an apples-to-apples comparison, we re-trained their AlexNet-based models on our data (graphs, material model, and illumination). Follow their paper, we restrict their prediction to the exposed graph parameters (and all parameters if no exposed parameters are defined).

As can be seen on the synthetic tests in Figure 5, our direct prediction is more accurate than the result from Hu et al., despite that our parameter prediction networks are more compact than theirs. We attribute these improvements to the use of the image-space perceptual loss in training the prediction networks, which is possible only because of our differentiable material graphs. Moreover, our optimization-based approach further improves over our direct prediction results, resulting in a significant gap in performance over their results.

These results are also consistent with the real-world captures shown in Figure 9. Here, we demonstrate that different variants of our optimization scheme, where we optimize different combinations of full or exposed parameters all outperform Hu et al. (see Sec. 6.4 for more discussions). During experimentation, we also find their network has a hard time disambiguating correlations between parameters due to the lack of knowledge of the material graph. For example, a graph consists of two exposed parameters both control the color of the diffuse albedo (one exposed parameter for RGB manipulation, the other for HSV manipulation), their network can get conflated, resulting in poor color reproduction (see Figure 9 red insets). This forces users to manually clean up the exposed parameters and prevents expanding their network to full parameter prediction for a majority of the procedural graphs.

Comparison with single-image per-pixel capture. In Figure 8, we compare our method to the state-of-the-art deep learning-based single-image SVBRDF capture methods of Deschaintre et al. [2018] (based on direct SVBRDF prediction) and Gao et al. [2019] (based on

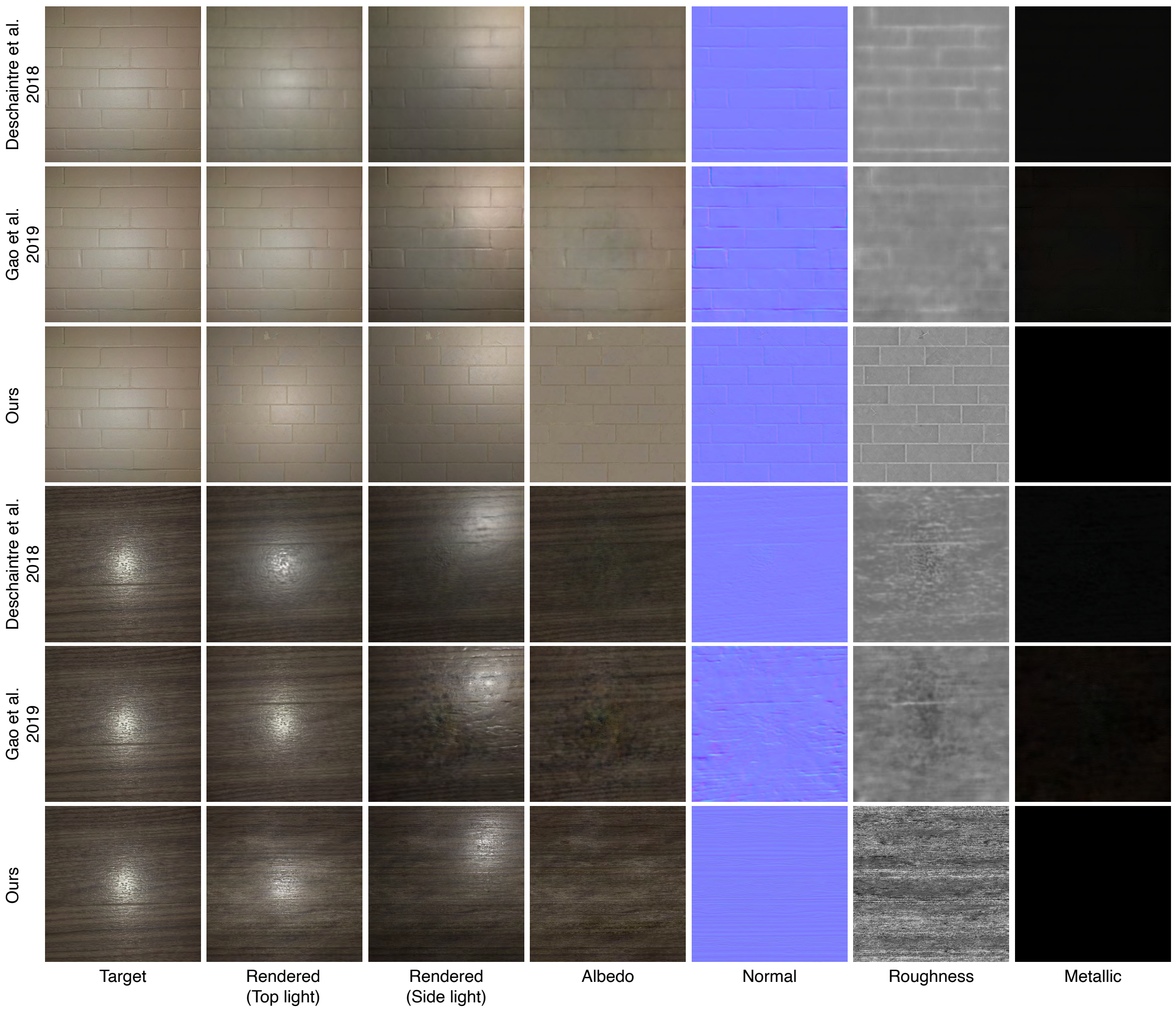

Fig. 8. Comparison of our method to single-image per-pixel SVBRDF capture methods of Deschaintre et al. [2018] and Gao et al. [2019] illustrate the key difference to our method. While the per-pixel methods are able to produce texture patterns more closely aligned to the exact target photo, they suffer from various artifacts such as blurring and color fringing. Our method, predicting node graph parameters instead of pixels directly, is much more robust to these problems, though of course at the cost of not matching the target texture patterns exactly.

optimization in an autoencoder latent space). These methods predict the per-pixel material maps that are well aligned to the target photo. However, because of the unconstrained nature of the problem, they suffer from noticeable artifacts such as blurring, color fringing, and degraded fidelity for novel light positions. These methods are also limited to the resolution and appearance of these results. By capturing into a procedural model, our method produces higher-quality materials that can be edited, tiled, and synthesized at arbitrary resolutions. This does come at the cost of not matching the target texture patterns exactly, but this is often not important in many material capture scenarios where users would want to capture the general appearance of the material, but not the exact structure of the input image. In conclusion, it is clear that the task of procedural material capture has (and will always have) its own trade-offs compared to the per-pixel material capture.

6.4 Additional Applications

Material Editing. A key advantage of procedural materials is their editability. Starting with the optimized results, artists can manually fine-tune the parameters to reproduce the exact material appearance they would like or sample around the optimized parameters to generate a family of similar textures. We demonstrate this second application in Figure 11, where we adjusted the node parameters

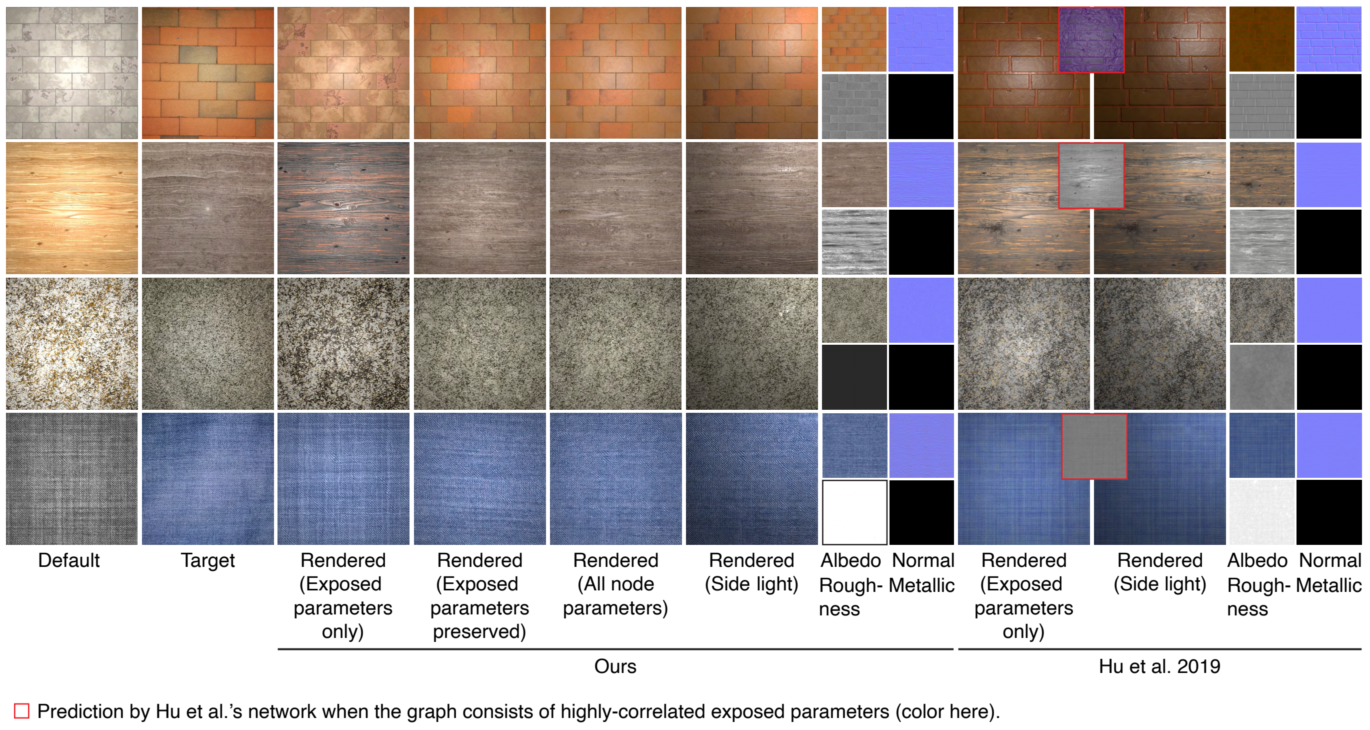
Default
Target
Rendered (Exposed parameters only)
Rendered (Exposed parameters preserved)
Rendered (All node parameters)
Rendered (Side light)
Albedo
Normal
Rough-ness
Metallic
Rendered (Exposed parameters only)
Rendered (Side light)
Albedo
Normal
Rough-ness
Metallic
Ours
Hu et al. 2019
🟥 Prediction by Hu et al.'s network when the graph consists of highly-correlated exposed parameters (color here).

Fig. 9. Comparison of our method to Hu et al. [2019] network prediction (without non-differentiable style transfer step) for real-world captured materials. For our results, we show optimization over exposed parameter only, exposed parameters with the rest of independent node parameters, and all node parameters. Our method overall produces a better match through optimizing a significantly larger design space and iteratively update the appearance to match the target. In contrast, the prediction by Hu et al. network is one-shot and can't not be iterative refined through back-propagation.

of the optimized materials to create four material variants for each example.

As mentioned before, some material graphs have exposed parameters that are specified by the artists who created these graphs. However, not every graph has exposed parameters, and as noted before optimizing only the exposed parameters can severely constrain the expressiveness of the graph. That said, exposed parameters can represent user-friendly controls over a graph's output. In Figure 9, we demonstrate the result of optimizing the exposed parameters of a graph to match a target image. In particular, we demonstrate that a hybrid approach where we optimize the exposed parameters and any parameters independent of the exposed parameters—called exposed parameters *preserved*—produces results visually comparable to full parameter optimization, while allowing for easy editability via the exposed parameters.

High-resolution Material Synthesis. Procedural materials are independent of resolution: once the parameters are optimized at a base resolution (512 × 512 in our results), we can synthesize high-resolution material maps using the same set of node parameters. In Figure 12, we picked 4 optimized, detail-rich materials and rendered their appearance at a resolution of 2048 × 2048 pixel (figures are cropped to fit the space). At higher resolutions, fine-grain details are revealed without loss of image sharpness. Such super-sampling would be very difficult to achieve with per-pixel material capture methods.

6.5 Limitations

MATch does not optimize the graph structure, thus its success ultimately relies on the expressiveness of the chosen graph. Consequently, it performs poorly or fails when unmodelled patterns are present in the input or the complexity of the input pattern exceeds the granularity of the deployed filters. For example, the optimized wood example in Figure 6 does not precisely reproduce the dot-like grooves in the normal map because the normal sub-graph cannot achieve this granularity, instead a similar visual effect is approximated through a noisy roughness map. A very similar example can also be found in Figure 10 second row. Figure 10 first row visualizes a more exaggerated example, where the substrate behind the input metal grate is completely unmodelled, and the optimization fails.

MATch currently does not optimize the generator nodes, thus unmatched input noise patterns result in poor reproduction. The optimized "Metal gritty" in Figure 6 is an example of this, where the ferrofluid pattern persists after the optimization despite the doughnut-like color gradient gets removed. More subtle examples can be found in "Amber leather" and "Red wall" of Figure 6. Figure 10 (third row) demonstrates a more exaggerated example, where a tile graph is used to reproduce a bed sheet with scattered square patterns.

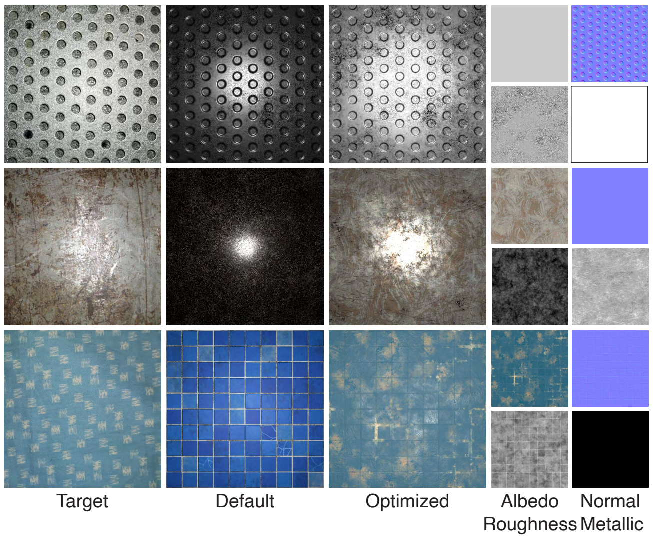

Fig. 10. Failure cases due to unmodeled objects (substrate behind the metal grate), limited expressiveness of overly simple graph structures (subgraphs that produce metallic of the metal grate, normal of the scratched metal, and the bed sheet example), and inability to optimize the generator nodes to match the input pattern (albedo of the scratched metal, the bed sheet example).

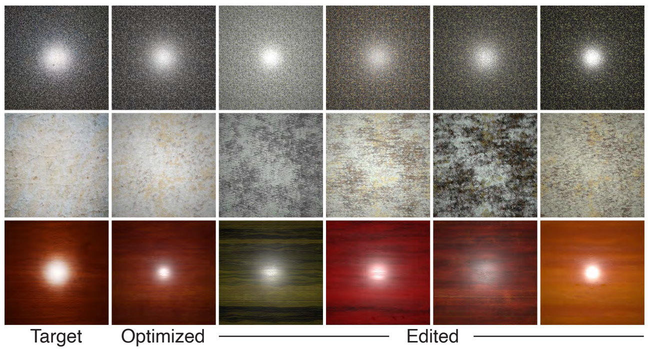

Fig. 11. Material editing. In each row, we show an optimized real-world material and variants generated by manually adjusting the optimized node parameters. Note that all the variants are generated from the same group of noise patterns, yet display a wide range of appearances.

Because the tile pattern is an input of the generator node, MATch can't relocate individual tiles, which remain visible in the final result.

Finally, MATch currently does not support full-fledged Fx-map Substance node and derived nodes such as tile generator and splatter. Thus, it is not able to convert graphs that use these nodes as intermediate nodes (only as generators). This makes conversion of certain types of materials such as grass difficult. This is a limitation of our current implementation, which does not occur in black-box approaches (e.g. Hu et al. [2019]) since differentiability is not a requirement.

In the future, the aforementioned shortcomings can be improved with further development of DiffMat and combining graph structure search with parameter optimization.

7 CONCLUSION AND FUTURE WORK

We have introduced MATch, a framework for matching the output appearance of a procedural material graph with respect to a user-input material image. MATch is enabled by our differentiable library DiffMat, which provides differentiable routines for procedural material building blocks (nodes) and automatic graph translation. Our framework allows direct optimization of material parameters using stochastic gradient descent methods, and further uses a neural network to select an appropriate procedural graph and initialize appropriate node parameters to accelerate and improve optimization. We have validated MATch's effectiveness on a large collection of synthetic and real-world materials.

In the future, we would like to further expand the capability of MATch. In terms of system input, we would like to go beyond the assumption of a single flat sample, and capture materials from images of curved objects, or with multiple materials in the image. In terms of material models, we would like to handle more complex material appearances including anisotropic BRDFs and spatially-varying specular highlights caused by wave optics based effects (i.e. glints due to diffraction and inference). In terms of DiffMat, we would like to fully support Fx-map and derived nodes such as tile generator and splatter; this will enable the inclusion of generator nodes into the optimization loop and also significantly expand the number of convertible material graphs. Finally, in this work we assumed that the material graph is given; an interesting direction is to synthesize the material graph from scratch for a set of user captured material photographs (ideally of one specific type of material). This is similar to the problem of neural architecture search (NAS) in deep learning that aims to automate the design of network architectures to maximize performance while minimizing computational cost. This will expand the space of material graphs that can be optimized and enable more intelligent and automatic procedural material capture.

ACKNOWLEDGMENTS

Parts of this work were done when Liang Shi was an intern at Adobe Research. This work is supported in part by the U.S. National Science Foundation (NSF) grant CHS-1815585. We thank Guo et al. [2019] for several PyTorch routines and discussions. We also thank Holly Rushmeier, Julie Dorsey and Yiwei Hu for sharing their SIGGRAPH Asia 2019 implementation.

REFERENCES

Adobe. 2019. Substance. https://docs.substance3d.com/sat.
Miika Aittala, Timo Aila, and Jaakko Lehtinen. 2016. Reflectance Modeling by Neural Texture Synthesis. *ACM Transactions on Graphics* 35, 4 (July 2016), 65:1–65:13.
Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2013. Practical SVBRDF Capture in the Frequency Domain. *ACM Transactions on Graphics* 32, 4 (July 2013), 110:1–110:12.
Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. 2015. Two-shot SVBRDF Capture for Stationary Materials. *ACM Transactions on Graphics* 34, 4 (July 2015), 110:1–110:13.
Brett Burley. 2012. Physically-based shading at Disney. In *ACM SIGGRAPH 2012 Courses.*
Valentin Deschaintre, Miika Aittala, Fredo Durand, George Drettakis, and Adrien Bousseau. 2018. Single-image SVBRDF Capture with a Rendering-aware Deep Network. *ACM Transactions on Graphics* 37, 4 (July 2018), 128:1–128:15.

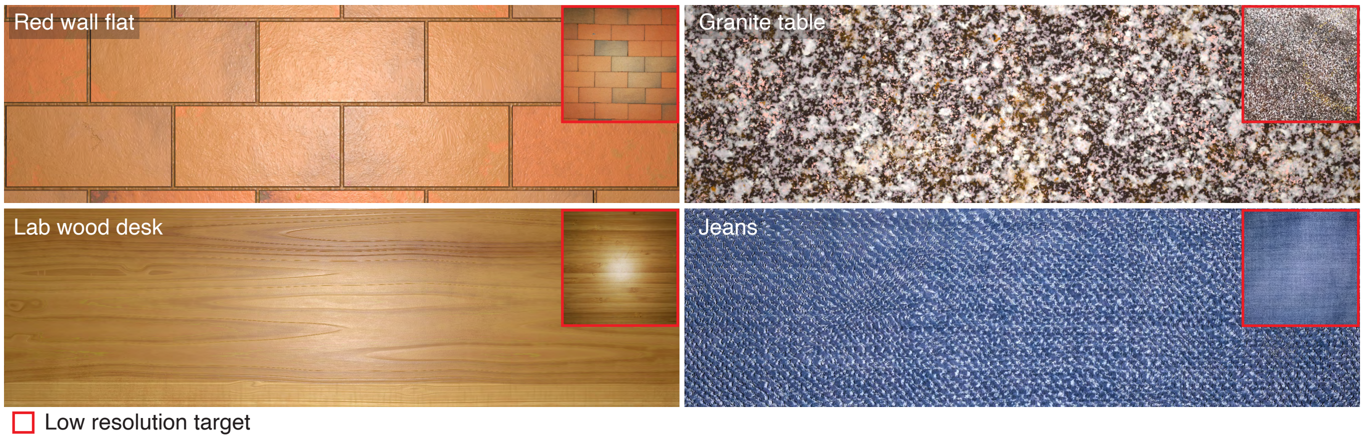

Fig. 12. High resolution procedural materials optimized for real-world photograph inputs. Readers are encouraged to zoom in and examine details.

Valentin Deschaintre, Miika Aittala, Frédo Durand, George Drettakis, and Adrien Bousseau. 2019. Flexible SVBRDF Capture with a Multi-Image Deep Network. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering)* 38, 4 (July 2019).

Valentin Deschaintre, George Drettakis, and Adrien Bousseau. 2020. Guided Fine-Tuning for Large-Scale Material Transfer. In *Computer Graphics Forum*, Vol. 39. Wiley Online Library, 91–105.

Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. 2012. Gabor Noise by Example. *ACM Transactions on Graphics* 31, 4 (July 2012), 9.

B. Galerne, A. Leclaire, and L. Moisan. 2017. Texton Noise. *Computer Graphics Forum* 36, 8 (2017), 205–218.

Duan Gao, Xiao Li, Yue Dong, Pieter Peers, Kun Xu, and Xin Tong. 2019. Deep Inverse Rendering for High-resolution SVBRDF Estimation from an Arbitrary Number of Images. *ACM Transactions on Graphics* 38, 4 (July 2019), 134:1–134:15.

Andrew Gardner, Chris Tchou, Tim Hawkins, and Paul Debevec. 2003. Linear Light Source Reflectometry. *ACM Transactions on Graphics* 22, 3 (July 2003), 749–758.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. 2015. A Neural Algorithm of Artistic Style. arXiv:cs.CV/1508.06576

L. A. Gatys, A. S. Ecker, and M. Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2414–2423. https://doi.org/10.1109/CVPR.2016.265

Dar'ya Guarnera, Giuseppe Claudio Guarnera, Abhijeet Ghosh, Cornelia Denk, and Mashhuda Glencross. 2016. BRDF Representation and Acquisition. *Computer Graphics Forum* (2016).

Yu Guo, Milos Hasan, Lingqi Yan, and Shuang Zhao. 2019. A Bayesian Inference Framework for Procedural Material Parameter Estimation. arXiv:cs.GR/1912.01067

Eric Heitz and Fabrice Neyret. 2018. High-Performance By-Example Noise Using a Histogram-Preserving Blending Operator. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 2 (Aug. 2018), 25.

Kato Hiroharu, Ushiku Yoshitaka, and Tatsuya Harada. 2018. Neural 3D Mesh Renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. 2020. DiffTaichi: Differentiable Programming for Physical Simulation. *ICLR* (2020).

Yiwei Hu, Julie Dorsey, and Holly Rushmeier. 2019. A Novel Framework for Inverse Procedural Texture Modeling. *ACM Transactions on Graphics* 38, 6 (Nov. 2019), 186:1–186:14.

Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. 2018. Exposure: A white-box photo post-processing framework. *ACM Transactions on Graphics* 37, 2 (July 2018), 1–17.

Diederik Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)* (12 2014).

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. 2018a. Differentiable Monte Carlo Ray Tracing through Edge Sampling. *ACM Transactions on Graphics* 37, 6 (July 2018), 222:1–222:11.

Zhengqin Li, Kalyan Sunkavalli, and Manmohan Chandraker. 2018b. Materials for masses: SVBRDF acquisition with a single mobile phone image. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 72–87.

Matthew M Loper and Michael J Black. 2014. OpenDR: An approximate differentiable renderer. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 154–169.

Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. 2019. Mitsuba 2: A Retargetable Forward and Inverse Renderer. *ACM Transactions on Graphics* 38, 6 (Nov. 2019), 203:2–203:17.

E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski. 2020. Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. https://arxiv.org/pdf/1910.02190.pdf

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR)*.

Ayush Tewari, Michael Zollhöfer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Pérez, and Christian Theobalt. 2017. MoFA: Model-based Deep Convolutional Face Autoencoder for Unsupervised Monocular Reconstruction. In *IEEE International Conference on Computer Vision (ICCV)*. 3735–3744.